Deadline Constrained Virtual Machine Provisioning and scheduling to improve QoS in Cloud Computing

[1] Sugney S. Kathrotia , [2] Prof. Rashmi S. Agrawal, [3] Suhradam M. Patel

[1]M.E.[Computer] Student, Department Of Computer Engineering, Atmiya Institute Of Technology And Science, Rajkot,Gujrat
[2] Professor, Department Of Computer Engineering, Atmiya Institute Of Technology And Science, Rajkot,Gujrat
[3] M.Tech.[Infotmation Technology] Student, Department Of Information Technology Engineering, Rajiv Gandhi Proudyogiki Vishwavidyalaya,Bhopal,Madhya Pradesh.

sugneykathrotia@gmail.com, rsagrawal@aits.edu.in, Suhradam@hotmail.com

***ABSTRACT*:** **Cloud Computing delivers IT resources as a Service over Internet. It is growing as the future for application deployment depending upon Quality of Service provided by the service provider. Virtualization provides the base for the same; it has certain pros like portability, efficient hardware utilization, ease in maintenance, cost benefit and cons like performance degradation if virtualized resources are not uniformly allocated. In this paper Tight Deadline request coming from user are managed. To meet Tight Deadline requirements we have created Heterogeneous Cloud Computing Environment (virtual machines with different powered processing elements). CloudSim toolkit is used to model the virtual machines over a node and other simulation based on deadline. This improves response time to user and request rejection ratio. Ultimately QoS (Quality of Service) gets improved.**

Keywords—Hetrogeneous Cloud Computing, Virtulization, Cloud Sim.

1: INTRODUCTION

Cloud computing made available a virtually infinite amount of resources hosted by public Cloud providers that charge for resource utilization in a pay-per-use model [1]. Public Cloud infrastructures can be combined with existing in-house resources from organizations in order to accelerate the execution of their distributed applications. This technique is called Cloud bursting, and the environment comprising such combined resources is termed Hybrid Cloud. When Cloud bursting is applied, the Resource Management System (RMS) coordinating the access to the resources has to determine when, how many, and for how long such resources are required and provision them dynamically. The RMS has also to determine which tasks will be executed on each resource and in which order (scheduling). A common approach to manage such access is to assign an allocation time where a user has exclusive access to a number of re-sources. More sophisticated resource managers such as Oracle (former Sun) Grid Engine [2] and Aneka [3] operate in a different mode where tasks that compose the application are queued and executed whenever there are free resources. Priority of tasks are periodically recalculated, what enables enforcement of organization-defined policies about access rights and Quality of Service (QoS) in the form of deadlines for application completion. Moreover, most approaches for dynamic provisioning operate in a per-job level, and thus they are inefficient because they fail in consider that other tasks could utilize idle cycles of Cloud resources. The latter aspect is especially relevant in the context of typical Infrastructure as a Service (IaaS) providers, which charge users in specific time intervals. To counter such lack of solutions for cost-effective dynamic provisioning and scheduling in hybrid Clouds, we present a coordinated dynamic provisioning and scheduling approach that is able to cost-effectively complete applications within their deadlines by considering the whole organization workload at individual tasks level when making decisions. The approach also contains an advanced accounting mechanism to determine the share of the cost of utilization of public Cloud resources to be assigned to each user.

2. RELATED WORK

The most of the existing scheduling policies for Clusters, Grids [4-10], and hybrid Clouds [11-13] either operate with user specification of allocation slots for utilization of resources or make decisions for a single job without considering jobs already queued. In the latter approach, decisions that optimize one job may cause delays to other jobs or, when Cloud resources are provisioned to complement local resources, may lead to underutilization of the extra resources. In the former approach, users are responsible for ensuring that the job can be executed Within the time slot. However, users typically overestimate their jobs' needs, what leads to inefficiencies in the scheduling process. Therefore, we apply a request model where users do not reserve resources during a time interval for job execution. Instead, users submit jobs and specify their deadlines (if any), and the scheduler submits tasks for execution on resources. The above model is also adopted by the Sun Grid Engine (SGE) [2] and the systems derived from it. Such systems over a scheduling policy for distributed jobs that allows priority to be assigned to users or groups. It also contains a model of deadline for job execution. However, in such a system deadline is defined in terms of start time of the job, whereas our model considers the completion time of the job. UniCloud1 is a software that However, their approach makes decision of whether using in-house resources or Cloud resources at job level (i.e., all the tasks that compose the job either run in-house or run on the Cloud) without reutilization of Cloud resources. Our approach, on the other hand, schedules at task level. This has the advantage of enabling a better utilization of Cloud resources by running tasks from other jobs if the billing interval is not over and the job that requested the Cloud resources finished. Dynamic provisioning of Cloud resources has been explored with different purposes. Such a system manages requests at single task level. Therefore, deadlines are determined for individual tasks, not for the whole job. It provisions resources.

3. Proposed Architecture

Our proposed RMS architecture is depicted in Figure 2. Requests for job execution are received by the Admission Control component. Accepted requests are received by the Scheduler component. Based on information about job queues, jobs' deadlines, and amount of available resources, requests for extra resources are sent from the Scheduler to the Provisioner. The Provisioner is responsible for acquiring resources from public Clouds and making them available to the Scheduler. Finally, the Accounting module interacts with the Scheduler to determine whether users have credit and authorization to request and use resources from public Clouds, and also to keep track of utilization of external resources so groups and users can be properly charged for public Cloud utilization. Fig. 3. Organization of resource pools and scheduling queues. The Admission Control accepts all the requests that do not have a deadline constraint. For requests with deadlines, it makes the decision whether the job can be accepted and completed within the deadline or the job must be rejected because it is unfeasible. To determine whether a request can be accepted, the Scheduler module is queried by the Admission Control module. The Scheduler than, considering user estimation, available resources, Cloud resources in use, and user access rights, replies to the Admission Control whether the user has permission and credit to run the job and whether the deadline is feasible or not. The Scheduler's reply is used as the final decision about job acceptance.

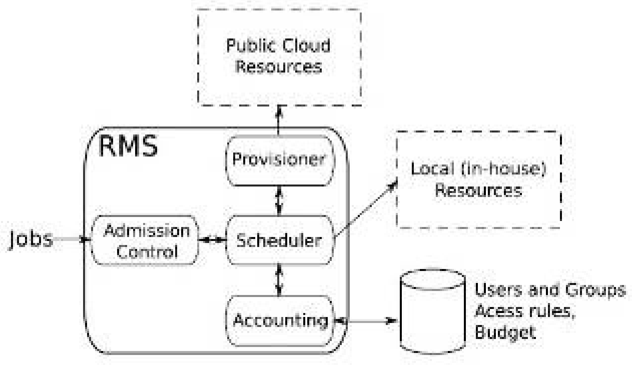

Fig.1 Proposed Resource Management System architecture for integrated dynamic provisioning and scheduling of applications in hybrid Clouds.

3.1 Scheduler

Jobs that are accepted by the Admission Control are received by the Scheduler module, which makes decisions based on a number of factors such as the pool to which the idle resources belongs to and job priority and ownership. In order to prevent starvation of regular jobs, a minimum amount of resources to be made available for regular tasks can be defined. These resources compose the regular pool and its access is coordinated via a regular queue. The rest of the local machines belong to the deadline pool, whose accesses are coordinated via deadline queues. Tasks that compose submitted jobs are forwarded either to the regular queue or to one of the deadline queues (there is one of such queues for each resource that belongs to the deadline pool). They respectively store tasks without deadline-constraints and tasks with such constraints. Tasks on each queue are rearranged every time a new job is received by the Scheduler and every time a task completes. A third set of queues, external is also present in the Scheduler. There is one of such queues for each user and it contains tasks that belong to jobs that require dynamic provisioning to complete before the deadline. Tasks on this queue execute preferentially in dynamically provisioned resources, as detailed later in this section.

Algorithm 1 details the procedure for building the regular queue. This procedure runs every time a new job is received and every time a new resource is added to this pool. The total time of each resource used by jobs from a group is summed up to give the total work wg of group gp (Lines 2 to 5). Groups are sorted in ascending order of Vi (Line 6), and each group receives a share of resources Ni that respects the amount of resources assigned to each group defined in the Scheduler (Line 8). The value Ng is the number of tasks from the group gp that go to the top of the queue. Ng tasks from the group with the lowest wi go to the top of the queue, followed by Nh tasks from the group with the second lowest Vi and so on, until all the shares are defined. The rest of the tasks are put in the end of the queue in arrival order (Line 11).Dispatching of tasks for execution depends on the pool that the idle resource belongs to. When a resource from the regular pool becomes idle, the task on top of the regular queue is dispatched for execution in such resource. If the regular queue is empty, the waiting task from deadline queues with the smallest lag time (which we define as the difference between the time to the deadline and the estimated execution time) is removed from its queue and dispatched for execution. Finally, if the deadline queue is also empty, the first task on the external queue is dispatched for execution.

```
Algorithm 1: Regular scheduler queue build up procedure.
Data: res: number of resources in the regular pool.
Data: max: maximum number of resources allowed for the group i.
1. Empty regular queue;
2. foreach group gp do
        Vi ← Vi/ ∑ j Vj, the proportional resource utilization by group gp during the current time window;
4 utilization List ← Vi;
5 end
6 sort utilization List in ascending order of wi;
7 foreach Vi in utilization List do
8 sharei ← min(maxi, [(1- wi)*rese];
9 add sharei tasks from group gp to the regular queue;
10 end
11 add remaining tasks in FIFO order to the regular queue;
```

3.2 Provisioner

The Provisioner makes decisions about utilization of public Cloud resources. It calculates the number of extra resources required to execute a job within its deadline and also decides if machines whose billing periods are _nishing will be kept for another period or not. The required number of resources is de_ned at task level: tasks that belong to an accepted job that can run in the deadline pool before the deadline are scheduled locally. Tasks that cannot be completed on time are put in the external queue by the scheduler, and provisioning decision is made based only on such tasks. Currently, the provisioner assumes a single type of VM to be provisioned. This increases the chance of successful allocation of Cloud resources because it enables acquisition of \reserved" or \pre-paid" resources. Most IaaS over such type of resource, which guarantees that, whenever resources are required, they will be available, as users paid for them upfront or via a premium plus discounted rates for utilization. Alternatively, the provisioner can register multiple providers, and use resources from another provider when the preferable one cannot supply the required resources. When a virtual machine is reaching the end of its billing period, the Provisioner decides whether the resource should be kept for the next billing period or if it should be decommissioned. This decision is based on the states of external deadline queues. The simplest case is when the external resource is idle or it is running a regular task. It happens when the other queues are empty. In this case, the resource is decommissioned by the Provisioner. A regular task running on the resource is rescheduled in the regular queue. If the provisioned resource is executing a deadline or external task, the resource is kept for the next billing period to avoid risk of missing the job's deadline.

In the case that the resource is no more necessary for the user that originally requested it, and there are still external tasks in the queue, the resource is reassigned for the user that needs the resource (providing it has authorization and credit to use them). In this case, accounting responsibilities for the reassigned resource is also changed, as detailed next.

4.Results and Discussion

Figure 4 presents results for utilization of public Clouds. The unit used is VM hours, which we de_ne as the sum of the wall clock time of each dynamically provisioned VM, from its creation to its destruction. Results show that our integrated provisioning approach was able to successfully reduce the utilization of public Cloud resources as a whole. Utilization of public Cloud resources was reduced to up to 20%. The smallest improvement generated by our integrated strategy was 5.2%, and the average reduction in public Cloud utilization was 10.24%. It represents a significant reduction in costs for organizations considering that our experiment simulated 1 day of resources utilization for 10 users. Because typical utilization scenarios are likely to be scaled to a bigger number of users for longer periods of time, the application of our approach can help organizations to significnatively reduce their budget of Cloud bursting. Paired t-tests on the Cloud utilization reported by different policies showed that task-level scheduling and provisioning caused 1% increase in the utilization of local resources (because of tasks that were kept locally instead of being sent to the Cloud). Because the total number of hours of the workload is the same, and the increased local load was smaller, we conclude that the significant reduction. In the number of Cloud resources is caused by a more effective utilization of such resources. This reduction in Cloud utilization happened without any impact in the capacity of our mechanism in meeting job deadlines. In fact, all the policies were able to meet deadline of 100% of jobs by applying a provisioning strategy. This is an expected effect as, because the system is able to provisioning as many public Cloud resources as necessary, and no unfeasible jobs were submitted, deadlines could always be met with a suffciently large amount of public Cloud resources.

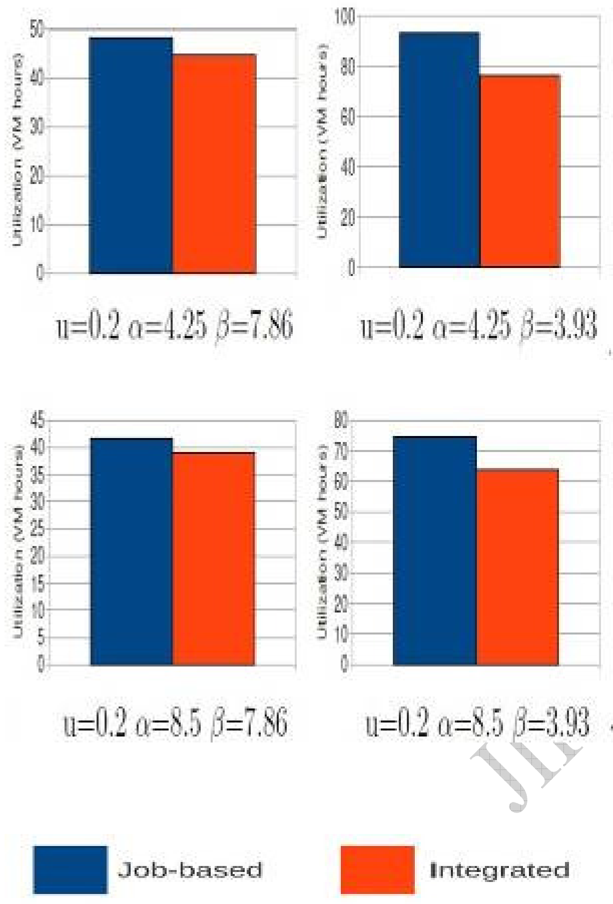

Fig.2 Public Cloud utilization in VM-hours for a 1-day load workload for different simulation scenarios. Job-based: traditional provisioning techniques applied at joblevel. Integrated: our integrated policy with task-level provisioning and reassignment of public resources. α and β denote the scale and shape of the job arrival distribution and u denotes the rate of urgent requests.

5. CONCLUSION AND FUTURE WORK

Cloud computing transformed the way that distributed applications are executed by making possible to complement in-house resources with pay-per use public Cloud resources. This makes possible for users to deadline a deadline for application execution and the budget to be spent, if necessary, for the deadline to be met. In this paper, we presented an architecture that enables Resource Management Systems to support the aforementioned tasks. We describe the architecture, a combined provisioning and scheduling strategy, and an approach for billing users for utilization of Cloud resources that compensates resources reallocated to other users when the deadline application completes before the end of resource billing period. Simulation experiments show that our approach makes an efficient utilization of public Cloud resources and enables deadlines to be met with reduced expenditure with public Cloud resources by organizations. As future research, we will investigate optimization strategies in order to enable better utilization of multicore resources, when they are available. We will also extend the algorithms to support workflows and other applications where the RMS has to consider dependencies between tasks during the scheduling.

REFRENCES

[1] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6) (Jun. 2009) 599-616

[2] Gentzsch, W.: Sun Grid Engine: towards creating a compute power grid. In: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, Australia (May 2001) 35-36.

[3] Vecchiola, C., Chu, X., Buyya, R.: Aneka: A software platform for .NET-based cloud computing. In Gentzsch, W., Grandinetti, L., Joubert, G., eds.: High Speed and Large Scale Scienti_c Computing. IOS Press, Amsterdam, The Netherlands (2009) 267-295.

[4] Feitelson, D.G.: Scheduling parallel jobs on clusters. In Buyya, R., ed.: High Performance Cluster Computing. Volume 1. Prentice-Hall, Upper Saddle River (1999).

[5] Braun, T.D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing 61(6) (Jun. 2001) 810-837

[6] Silva, D., Cirne, W., Brasileiro, F.: Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In Kosch, H.Boszorm_enyi, L., Hellwagner, H., eds.: Euro-Par 2003 Parallel Processing.Volume 2790 of Lecture Notes in Computer Science., Springer (Aug. 2003) 169-180

[7] Cooper, K., et al.: New grid scheduling and rescheduling methods in the GrADS project. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, USA (Apr. 2004)

[8] Weng, C., Lu, X.: Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid. Future Generation Computer Systems 21(2) (Feb. 2005) 271-280

[9] Dong, F.:A taxonomy of task scheduling algorithms in the grid. Parallel Processing Letters 17(4) (Dec. 2007) 439-454
[10] Salehi, M.A., Javadi, B., Buyya, R.: Resource provisioning based on lease preemption in InterGrid. In: Proceedings of the 34th Australasian Computer Science Conference (ACSC'11), Perth, Australia (Jan. 2011).
[11] Assun_c~ao, M.D., di Costanzo, A., Buyya, R.: Evaluating the cost-bene_t of using cloud computing to extend the capacity of clusters. In: Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC'09), Munich, Germany (Jun. 2009) 141-150.
[12] Salehi, M., Buyya, R.: Adapting market-oriented scheduling policies for cloud computing. In Hsu, C.H., Yang, L., Park, J., Yeo, S.S., eds.: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10). Volume 6081 of Lecture Notes in Computer Science., Busan, South Korea, Springer (May 2010) 351-362.
[13] Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Multicloud deployment of computing clusters for loosely coupled MTC applications. IEEE Transactions on Parallel and Distributed Systems 22(6) (Jun. 2011) 924-930